236371 Project in Distributed Programming – Serverless Classifier on OpenWhisk

Programmer's Guide

Introduction

In this project we implemented a serverless image classifier using OpenWhisk. The classifier is accessible through a custom website where the user can upload an image that they want to be classified. The website encodes the image and sends it to the serverless classifier. Once the image is classified, the result is returned to the user's page and displayed there. Consult the User Manual for instructions on how to use the classifier.

Specification

The architecture of the solution and functional specification can be seen in the Functional Specification document. It specifies the design decisions we made concerning the project, and how we planned on implementing it. It also includes dependencies and links to some documentation websites we relied on while implementing the project.

Setup

Local Installations

All the necessary code and documentation can be found on the github repository: https://github.com/amitaifrey/serverless-classifier. As you can see in the files, you need to have Python 3.7 and Docker installed on your PC.

To create your own environment to run the project (or a similar one), first you must create an IBM cloud account and a Docker Hub account. More information about these services can be found at https://www.ibm.com/cloud and at https://hub.docker.com.

Once you have completed installing and creating all the accounts, download the image recognition model you want to use. This project uses NASNetMobile, and you can download it by simply running a python interpreter and importing the model – it will be downloaded to the default Keras models location. If you want to use a different model, download it, and change the python code in action.py to use your model.

After that, create a docker image using the Dockerfile in the project directory. This may take a while because it downloads all the necessary python libraries to the docker. It also copies the model you downloaded into the docker, so make sure you downloaded it to the correct path. Once the docker image is done, upload it to Docker Hub so you will be able to use it in the IBM Cloud environment.

Creating a Cloud Function

Once you have created the docker image and an IBM cloud account, log in to the console and go to the Cloud Functions section. Follow the instructions on the page to create a new namespace, that will contain your cloud function. I recommend using the IBM Cloud CLI because it is straightforward, has all the options and we will use it later as well. I have tested it on both Windows and Linux, and I assume it also works well on Mac. More information can be found at: https://cloud.ibm.com/functions/learn/cli.

Once you have created the namespace, it is time to create the cloud function:

- CLI: Run the following line from a console:
  ibmcloud fn action create classify --docker <docker path on dockerhub> action.py --web true --memory 512
- Web: Sadly, IBM doesn't currently support creating a cloud function with a custom docker using the web interface. I included this option here because it might be possible in the future, so I recommend checking out the web interface to see if something changed.

Notice that we enable the web option to allow us to connect a web API to the function. Additionally, we set memory to 512MB. This is done because the image recognition model we are using requires at least this much RAM to run effectively. If you plan on using different models or functions, you might need to change this parameter. We also specify the python script the function will run, so if you want to make any changes to it you should do so before creating the function.

If you want to change anything in the function such as its name or the python script, you can simply delete it and create it again, or modify it – all using the CLI. The specific CLI commands appear in the scripts.sh file in the project's directory.

Now that the function has been created, we can invoke it using the CLI to check that it works. First, create a JSON file with a dictionary that has a "image" key and the value contains a base64 encoded JPEG – an example is also included in the project files. Then run the following command: ibmcloud fn action invoke --result classify --param-file params.json and wait for the result.

Web API

Once this invocation works, we can continue to the next step – connecting the cloud function to a web API. This can be done by following the instructions at https://cloud.ibm.com/functions/apimanagement. Once you are done configuring the web API, we can invoke the function from an HTTP call using the following command (on Linux):
curl -v -u <username>:<password> -d @params.json -H "Content-Type: application/json" -X POST https://eu-gb.functions.cloud.ibm.com/api/v1/namespaces/<namespace>/actions/classify?blocking=true and receive the results as an HTTP response. This is used in the website we will now deploy.

Website on S3 buckets

To deploy a website on an S3 bucket, we first upload the files to the Cloud Object Storage service on the IBM cloud. All the necessary files are included in the project directory – they are the .html .js and .css files. Modify the .js file to use your Cloud Function API (and not mine!), then upload them to a location that you later want to connect to your web/DNS service.

Once they are uploaded and have a correct path, you can simply browse that location and the html files imports all the other files. Follow the instructions in the User Manual to run the classifier from the website.

Adding new features

This project is very basic and was built to get familiar with the environment and present a simple example on how to use it. To add new features or modify the existing project, you are more than welcome to fork the repository on Github and change whatever you see fit. Most of the simpler changes can be done by changing a few lines of code or configuring the service a little different. More complex changes will probably require some more work.

Documentation on the code itself and the CLI commands we used can be found in their respective files. Additional documentation on libraries or services we used can be found online, like documentation of TensorFlow or the IBM Cloud services.